1 Objectives

Recursion is one of the most powerful ideas in Computer Science and in Mathematics. A proper understanding of it is essential. All the lectures from here on out will really be about recursion.

- Understand how recursion is related to induction
- Know four patterns of recursion:
  Iterating, Mapping, Folding, Accumulating
- Know how to determine the time complexity of a recursive operation
- Know how to turn a linear recursion into an exponential recursion

2 Examples

2.1 Mapping Recursion

```
# let rec doubleList lst = match lst with
  | [] -> []
  | x::xs -> 2 * x :: doubleList xs;;
val doubleList : int list -> int list = <fun>
# doubleList [4;6;8];;
- : int list = [8; 12; 16]
```

2.2 Folding Recursion

Another common form "folds" a list via some function.

```
# let rec multList lst = match lst with
  | [] -> 1
  | x::xs -> x * multList xs;;
val multList : int list -> int = <fun>
# multList [2;4;6];;
- : int = 48
```

This computes $(2 * (4 * (6 * 1)))$.

3 Problems

Here are some example problems which will be useful to study in preparation for the exam. Some of these may be done in class.

1. Write an OCaml function that returns the maximum element of a list. Use forward recursion. (Assume the list always has at least 1 element.)
2. Now write the same function, but use tail recursion.
3. What is the running time of the following function?

```
# let rec doubleSum lst = match lst with
  | [] -> 0
  | x::xs -> doubleSum xs + doubleSum xs;;
val doubleSum : 'a list -> int = <fun>
```

4. Fix the above function to make it run in linear time.
5. What is a sub-expression?
6. What does it mean when an expression is in *tail position*?
7. What is a tail call?
8. What is the advantage of tail recursion over non-tail recursion?
9. What is iterating recursion? Write an example function.
10. What is mapping recursion? Write an example function.
11. What is folding recursion? Write an example function.
12. Convert the `doubleList` function above into tail recursive form.
13. Convert the `multList` function above into tail recursive form.